ELSEVIER

Theoretical Computer Science 293 (2003) 243–259

Theoretical Computer Science

www.elsevier.com/locate/tcs

# A note on the strong and weak generative powers of formal systems

Aravind K. Joshi *, [1]

*Department of Computer and Information Science, and Institute for Research in Cognitive Science, University of Pennsylvania, Room 555 Moore School, Philadelphia, PA 19104, USA*

**Abstract**

This paper is a note on some relationships between the strong and weak generative powers of formal systems, in particular, from the point of view of squeezing more strong power out of a formal system without increasing its weak generative power. We will comment on some old and new results from this perspective. Our main goal of this note is to comment on the strong generative power of context-free grammars, lexicalized tree-adjoining grammars (and some of their variants) and Lambek grammars, especially in the context of crossing dependencies, in view of the recent work of Tiede (Ph.D. Dissertation, Indiana University, Bloomington, 1999).


*Keywords:* Crossing dependencies; Lambek grammars; Lexicalized tree-adjoining grammars; Multi-component grammars; Strong generative capacity; Tree insertion grammars

## 1. Introduction

Strong generative power (SGP) relates to the set of structural descriptions (derivation trees, directed acyclic graphs, proof trees, etc.) which a formal system is capable of assigning to the strings it specifies. Weak generative power (WGP) refers to the set of strings characterized by the formal system. SGP is clearly the primary object of interest from the linguistic point of view. WGP is often used to locate a formal system within one or another hierarchy of formal grammars. Clearly, a study of the relationship between WGP and SGP is highly relevant, both formally and linguistically. Although almost from the beginning of the work in mathematical linguistics, there has been interest in the study of this relationship, the results are few, as this relationship

* Tel.: +1-215-898-8540; fax: +1-215-898-0587.
*E-mail address:* joshi@linc.cis.upenn.edu (A.K. Joshi).
[1] This work was partially supported by NSF Grant SBR8920230.


PII: S0304-3975(01)00347-4

is quite complex and not always easy to study mathematically (see [13] for a recent comprehensive discussion of SGP).

Our main goal in this note is to comment on the SGP of context-free grammars (CFGs), lexicalized tree-adjoining grammars (and some of their variants) and Lambek grammars, especially in the context of crossing dependencies, in view of the recent work of Tiede [21].

## 2. Context-free grammars

McCawley [12] was the first person to point out that the use of context-sensitive rules by linguists was really for checking structural descriptions (thus related to SGP) and not for characterizing strings (i.e., WGP), suggesting that this use of context-sensitive rules possibly does not give more WGP than CFGs. Peters and Ritchie [15] showed that this was indeed the case. These results are closely related to the notion of recognizable sets of trees as explained below.

In a CFG, $G$ the derivation trees of $G$ correspond to the possible structural descriptions assignable by $G$. It is easily shown that there are tree sets whose yield language is context-free but the tree sets are not the tree sets of any CFG.

$T$ in Fig. 1 is not a set of derivation trees for any CFG. Clearly, in any CFG $G$ the rules for $A$ will get mixed up and there will be no way we can make sure that all $a$'s are on the left and all $b$'s are on the right. The string language is, of course, $\{a^n b^m | m, n > 1\}$, which is a context-free language. What is the relationship between the trees of the CFG corresponding to this language and the set $T$? Thatcher [20] showed that the relationship is very close. Sets such as $T$, called recognizable sets, are the same as the tree sets of CFGs except possibly for relabeling. In Fig. 1 if the $A$'s on the right-hand side are labeled by $B$'s then it is easily seen that the new tree set can be easily generated by a CFG. It turns out that the tree sets 'analyzable' (i.e., checkable) by context-sensitive rules, as suggested by McCawley are indeed recognizable sets.

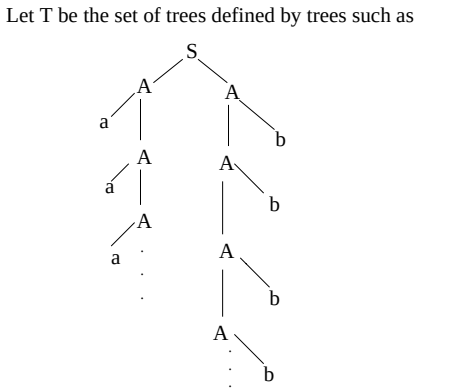



Fig. 1. A recognizable set of trees.

Thatcher's result shows that this notion of 'locality' can be captured by finite state tree automata. Later Joshi et al. [4] and Rogers [16] showed that the notion of 'local context' (or local tree domains) can be made substantially richer yet maintaining characterizability by finite state tree automata. These results can be interpreted as attempts to squeeze more strong power out of a formal system without increasing WGP beyond CFGs. [2]

## 3. Lexicalized tree-adjoining grammars (LTAG)

LTAGs were motivated by several considerations, one of which is directly relevant to the topic of this paper. This motivation is concerned with assigning a crossing dependencies type of structural description to a language such as $\{a^n b^n | n \geqslant 1\}$; which is a context-free language. Such a description is not obtainable by a CFG. It is possible to assign such a structural description by an LTAG as will be shown later (see Fig. 17). LTAGs are more powerful than CFGs but only slightly. LTAGs and related systems are characterized as mildly context sensitive grammars. Several variants of LTAG have been studied. Two of these variants are directly relevant to the main points of this note. One variant (tree-local multi-component LTAG, MC-LTAG) is concerned with augmenting the SGP of LTAGs without increasing their WGP. The other variant (tree insertion grammar, TIG) is concerned with adding some restrictions to LTAG such that the WGP of the system is the same as that of CFGs, however, the SGP is more than that of CFGs. These grammars are relevant to the Lambek grammars (LG) as LGs are weakly equivalent to CFGs but they have more SGP than CFGs.

LTAG is a formal tree rewriting system. LTAGs have been extensively studied both with respect to their formal properties and their linguistic relevance. The motivations for the study of LTAG are both linguistic and formal. The elementary objects manipulated by LTAG are structured objects (trees or directed acyclic graphs) and not strings. Using structured objects as the elementary objects of the formal system, it is possible to construct formalisms whose properties relate directly to the study of strong generative capacity (SGP), which is more relevant to the linguistic descriptions than the weak generative capacity (WGP).

Each grammar formalism specifies a domain of locality, i.e., a domain over which various dependencies (syntactic and semantic) can be specified. It turns out that the various properties of a formalism (syntactic, semantic, computational) follow, to a large extent, from the initial specification of the domain of locality.

[2] We must point out here the work of Uwe Mönnich and his colleagues which bear on the issues of strong and weak generative capacities. They have investigated the algebraic technique of 'lifting' to code executable information into context-free trees, which are later unpacked into the intended trees, covering the non-context-free yields. This work appears in this issue [8].

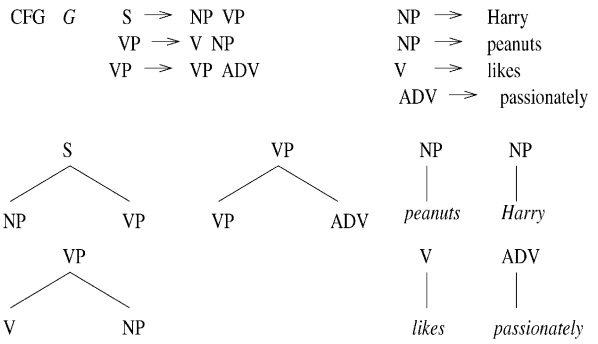



Fig. 2. Domain of locality of a context-free grammar.

### 3.1. Domain of locality of CFGs

In a context-free grammar (CFG) the domain of locality is the one-level tree corresponding to a rule in a CFG (Fig. 2). It is easily seen that the arguments of a predicate (for example, the two arguments of *likes*) are not in the same local domain. The two arguments are distributed over the two rules (two domains of locality) — $S \rightarrow NP\ VP$ and $VP \rightarrow V\ NP$. They can be brought together by introducing a rule $S \rightarrow NP\ V\ VP$. However, then the structure provided by the $VP$ node is lost. We should also note here that not every rule (domain) in the CFG in (Fig. 2) is lexicalized. The three rules on the right are lexicalized, i.e., they have a lexical anchor. The rules on the left are not lexicalized. The second and the third rules on the left are almost lexicalized, in the sense that they each have a preterminal category ($V$ in the second rule and $ADV$ in the third rule), i.e., by replacing $V$ by *likes* and $ADV$ by *passionately* these two rules will become lexicalized. However, the first rule on the left ($S \rightarrow NP\ VP$) cannot be lexicalized. Can a CFG be lexicalized, i.e., given a CFG, $G$, can we construct another CFG, $G'$, such that every rule in $G'$ is lexicalized and $T(G)$, the set of (sentential) trees (i.e., the tree language of $G$) is the same as the tree language $T(G')$ of $G'$? It can be shown that this is not the case [5]. Of course, if we require that only the string languages of $G$ and $G'$ be the same (i.e., they are weakly equivalent) then any CFG can be lexicalized. This follows from the fact that any CFG can be put in the Greibach normal form where each rule is of the form $A \rightarrow w\,B1\,B2 \cdots Bn$ where $w$ is a lexical item and the $B$'s are non-terminals. The lexicalization we are interested here requires the tree languages to be the same, i.e., we are interested in the 'strong' lexicalization. To summarize, a CFG cannot be strongly lexicalized by a CFG. This follows from the fact that the domain of locality of CFG is a one-level tree corresponding to a rule in the grammar. Note that here we are concerned with two issues — (1) lexicalization of each elementary domain and (2) the encapsulation of the arguments of the lexical anchor in the elementary domain of locality. The second issue is independent of the first issue. From a mathematical point of view the first issue, i.e., the lexicalization of the elementary domains of locality is the crucial one. We can obtain strong lexicalization without satisfying the requirement that the arguments of the lexical anchor are

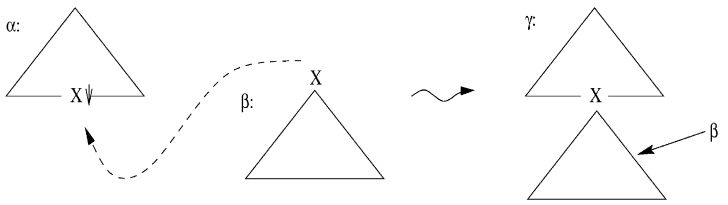



Fig. 3. Substitution.

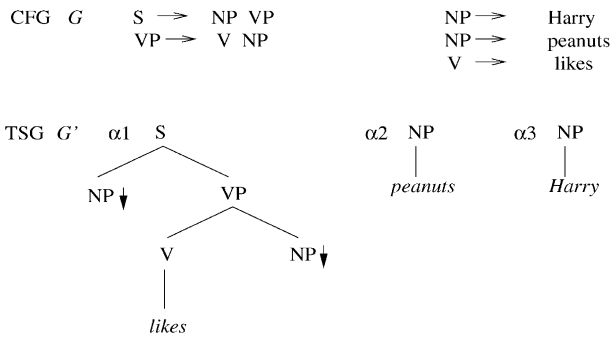



Fig. 4. Tree substitution grammar.

encapsulated in the elementary domain. Of course, from the linguistic point of view this encapsulation is very crucial. What this means is that among all possible strong lexicalizations we should choose only those that meet the requirements of encapsulation. For our discussions in this note we will assume that we always make such a choice.

### 3.2. Lexicalization of CFGs

Now we can ask the following question. Can we strongly lexicalize a CFG by a grammar with a larger domain of locality? Figs. 3 and 4 show a tree substitution grammar (TSG) where the elementary objects (building blocks) are the three trees in Fig. 4 and the combining operation is the tree substitution operation shown in Fig. 3. Note that each tree in the TSG, $G'$ is lexicalized, i.e., it has a lexical anchor. It is easily seen that $G'$ indeed strongly lexicalizes $G$. However, TSGs fail to strongly lexicalize CFGs in general. We show this by an example. Consider the CFG, $G$, in Fig. 5 and a proposed TSG, $G'$. It is easily seen that although $G$ and $G'$ are weakly equivalent they are not strongly equivalent. In $G'$, suppose we start with the tree $\alpha_1$ then by repeated substitutions of trees in $G'$ (a node marked with a vertical arrow denotes a substitution site) we can grow the right-hand side of $\alpha_1$ as much as we want but we cannot grow the left-hand side. Similarly for $\alpha_2$ we can grow the left-hand side as much as we want but not the right-hand side. However, trees in $G$ can grow on both sides. Hence, the TSG, $G'$, cannot strongly lexicalize the CFG, $G$ [5].

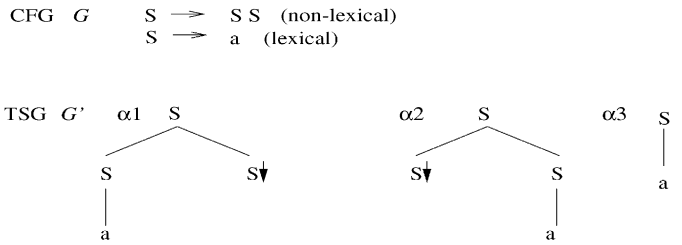



Fig. 5. A tree substitution grammar.

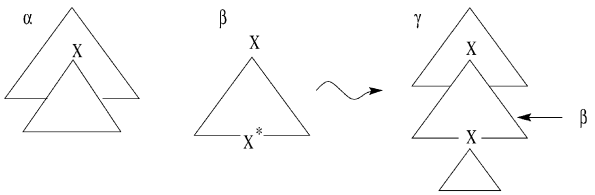



Fig. 6. Adjoining.

We now introduce a new operation called 'adjoining' as shown in Fig. 6. Adjoining involves splicing (inserting) one tree into another. More specifically, a tree $\beta$ as shown in Fig. 6 is inserted (adjoined) into the tree $\alpha$ at the node $X$ resulting in the tree $\gamma$. The tree $\beta$, called an auxiliary tree, has a special form. The root node is labeled with a non-terminal, say $X$ and on the frontier there is also a node labeled $X$ called the foot node (marked with $^*$). There could be other nodes (terminal or non-terminal) on the frontier of $\beta$. These non-terminal nodes will be marked as substitution sites (with a vertical arrow). Thus if there is another occurrence of $X$ (other than the foot node marked with $^*$) on the frontier of $\beta$ it will be marked with the vertical arrow and that will be a substitution site. Given this specification, adjoining $\beta$ to $\alpha$ at the node $X$ in $\alpha$ is uniquely defined. Adjoining can also be seen as a pair of substitutions as follows: The subtree at $X$ in $\alpha$ is detached, $\beta$ is substituted at $X$ and the detached subtree is then substituted at the foot node of $\beta$. A tree substitution grammar when augmented with the adjoining operation is called a tree-adjoining grammar (lexicalized tree-adjoining grammar because each elementary tree is lexically anchored). In short, LTAG consists of a finite set of elementary trees, each lexicalized with at least one lexical anchor. The elementary trees are either initial or auxiliary trees. Auxiliary trees have been defined already. Initial trees are those for which all non-terminal nodes on the frontier are substitution nodes. It can be shown that any CFG can be strongly lexicalized by an LTAG [5].

In Fig. 7 we show a TSG, $G'$, augmented by the operation of adjoining, which strongly lexicalizes the CFG, $G$. Note that this LTAG looks the same as the TSG considered in Fig. 5. However, now trees $\alpha_1$ and $\alpha_2$ are auxiliary trees (the foot nodes marked with $*$) that can participate in adjoining. Since adjoining can insert a tree in the interior of another tree it is possible to grow both sides of the tree $\alpha_1$ and tree $\alpha_2$,

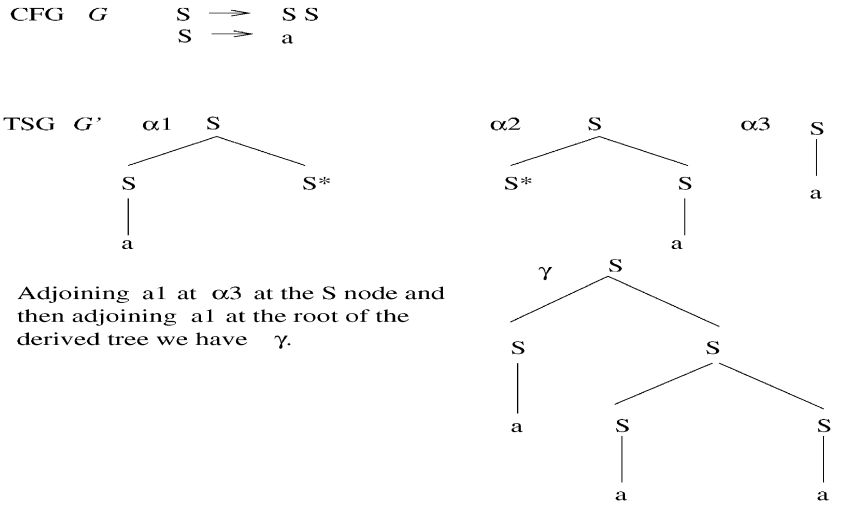



Fig. 7. Adjoining arises out of lexicalization.

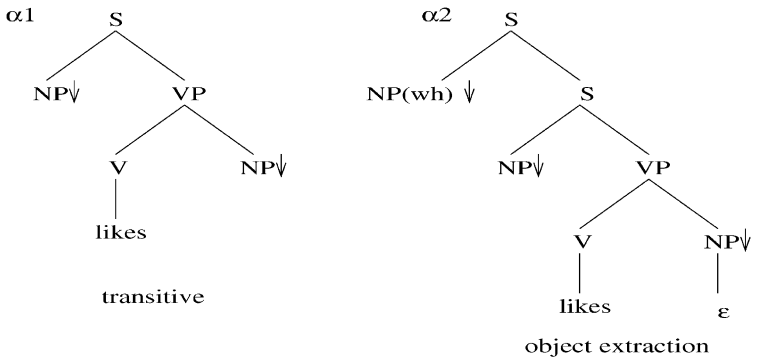



Fig. 8. LTAG: elementary trees for *likes*.

which was not possible earlier with substitution alone. In summary, we have shown that by increasing the domain of locality we have achieved the following: (1) lexicalized each elementary domain, (2) introduced an operation of adjoining, which would not be possible without the increased domain of locality (note that with one-level trees as elementary domains adjoining becomes the same as substitution since there are no interior nodes to be operated upon), and (3) achieved strong lexicalization of CFGs.

### *3.3. Lexicalized tree-adjoining grammars*

Rather than giving formal definitions for LTAG and derivations in LTAG we will give a simple example to illustrate some key aspects of LTAG. We show some elementary trees of a toy LTAG grammar of English. Fig. 8 shows two elementary trees for a verb such as *likes*. The tree $\alpha_1$ is anchored on *likes* and encapsulates the two

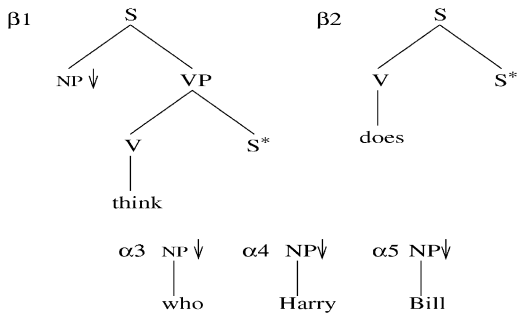



Fig. 9. LTAG: sample elementary trees.

arguments of the verb. The tree $\alpha_2$ corresponds to the object extraction construction. Since we need to encapsulate all the arguments of the verb in each elementary tree for *likes*, for the object extraction construction, for example, we need to make the elementary tree associated with *likes* large enough so that the extracted argument is in the same elementary domain. Thus, in principle, for each 'minimal' construction in which *likes* can appear (for example, subject extraction, topicalization, subject relative, object relative, passive, etc.) there will be an elementary tree associated with that construction. By 'minimal' we mean when all recursion has been factored away. This factoring of recursion away from the domain over which the dependencies have to be specified is a crucial aspect of LTAGs as they are used in linguistic descriptions. This factoring allows all dependencies to be localized in the elementary domains. In this sense, there will, therefore, be no long distance dependencies as such. They will all be local and will become long distance on account of the composition operations, especially adjoining.

Fig. 9 shows some additional trees. Trees $\alpha_3$, $\alpha_4$, and $\alpha_5$ are initial trees and trees $\beta_1$ and $\beta_2$ are auxiliary trees with foot nodes marked with $*$. A derivation using the trees in Figs. 8 and 9 is shown in Fig. 10. The trees for *who* and *Harry* are substituted in the tree for *likes* at the respective *NP* nodes, the tree for *Bill* is substituted in the tree for *think* at the *NP* node, the tree for *does* is adjoined to the root node of the tree for *think* tree (adjoining at the root node is a special case of adjoining), and finally the derived auxiliary tree (after adjoining $\beta_2$ to $\beta_1$) is adjoined to the indicated interior *S* node of the tree $\alpha_2$. This derivation results in the derived tree for *who does Bill think Harry likes* as shown in Fig. 11. Note that the dependency between *who* and the complement *NP* in $\alpha_2$ (local to that tree) has been stretched in the derived tree in Fig. 11. This tree is the conventional tree associated with this sentence.

However, in LTAG there is also a derivation tree, i.e., the tree that records the history of composition of the elementary trees associated with the lexical items in the sentence. This derivation tree is shown in Fig. 12. The nodes of the tree are labeled by the tree labels such as $\alpha_2$ together with the lexical anchor. The derivation tree is the crucial derivation structure for LTAG. We can obviously build the derived tree from the derivation tree. For semantic computation the derivation tree (and not the derived tree)

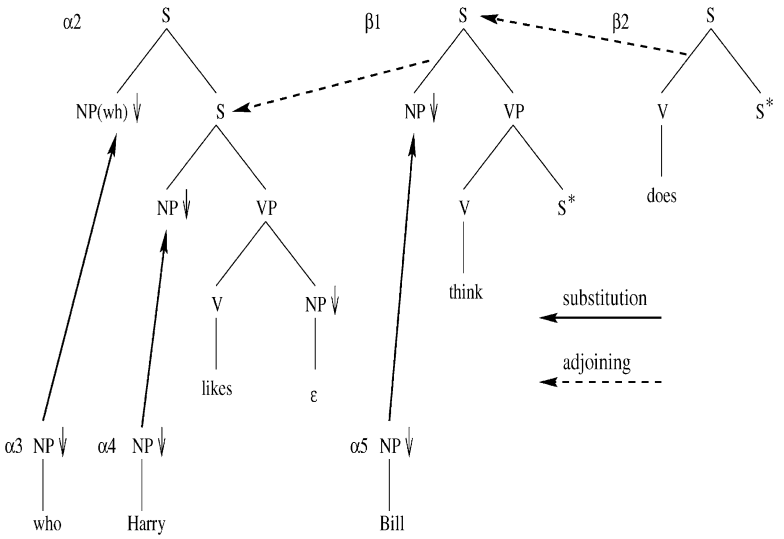



Fig. 10. LTAG derivation for *who does Bill think Harry likes.*

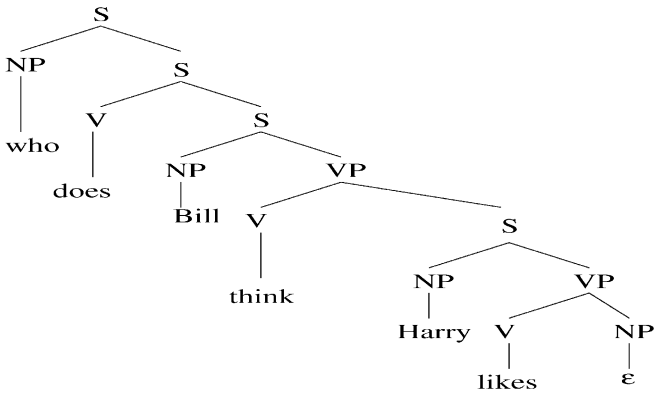



Fig. 11. LTAG derived tree for *who does Bill think Harry likes.*

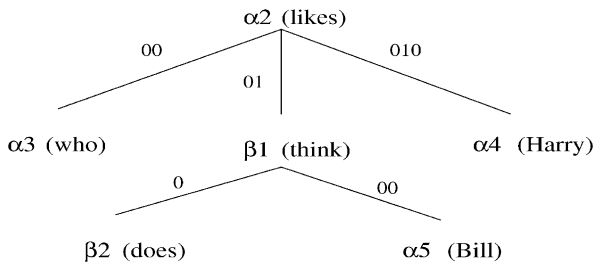



Fig. 12. LTAG derivation tree.

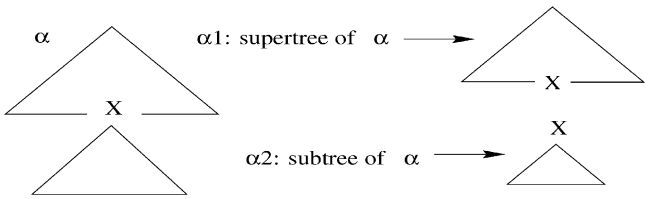



Fig. 13. Adjoining as Wrapping 1.

is the crucial object. Compositional semantics is defined on the derivation tree. The idea is that for each elementary tree there is a semantic representation associated with it and these representations are composed using the derivation tree. Since the semantic representation for each elementary tree is directly associated with the tree there is no need to reproduce necessarily the internal hierarchy in the elementary tree in the semantic representation [6]. This allows the so-called 'flat' semantic representation. It also helps in dealing with some non-compositional aspects as in the case of rigid and flexible idioms.

The two key properties of LTAG are (1) the extended domain of locality (EDL) (as compared to CFG), which allows (2) factoring recursion from the domain of dependencies (FRD), thus making all dependencies local. All other properties of LTAG (mathematical, linguistic, and computational) follow from EDL and FRD. LTAGs belong to the so-called class of mildly context-sensitive grammars [3]. Context-free languages (CFLs) are properly contained in the class of languages of LTAG, which in turn are properly contained in the class of context-sensitive languages.

## 4. Some variants of LTAG

### *4.1. Multi-component LTAG (MC-LTAG)*

MC-LTAG can be motivated by taking an alternate perspective on adjoining. This perspective also shows how a particular class of MC-LTAG, the so-called Tree-local MC-LTAGs increase SGP of LTAGs without increasing WGP.

In adjoining we insert an auxiliary tree, say, with root and foot nodes labeled with $X$, in a tree at a node with label $X$. In Figs. 13 and 14 we present an alternate perspective on adjoining. The tree $\alpha$ which receives adjunction at $X$ can be viewed as made up of two trees, the supertree at $X$ and the subtree at $X$ as shown in Fig. 13. Now, instead of the auxiliary tree $\beta$ adjoined to the tree $\alpha$ at $X$ we can view this composition as a wrapping operation — the supertree of $\alpha$ and the subtree of $\alpha$ are wrapped around the auxiliary tree $\beta$ as shown in Fig. 14. The resulting tree $\gamma$ is the same as before. Wrapping of the supertree at the root node of $\beta$ is like adjoining at the root (a special case of adjoining) and the wrapping of the subtree at the foot node of $\beta$ is like substitution. Hence, this wrapping operation can be described in terms of substitution and adjoining. This is clearly seen in the linguistic example in Figs. 15

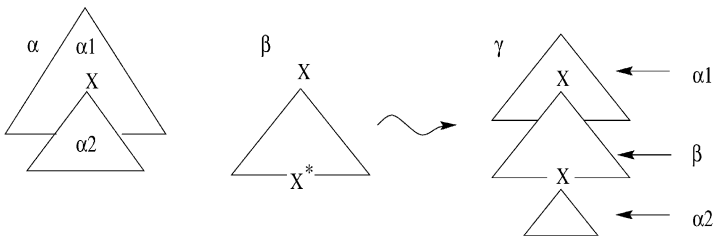



Fig. 14. Adjoining as Wrapping 2.

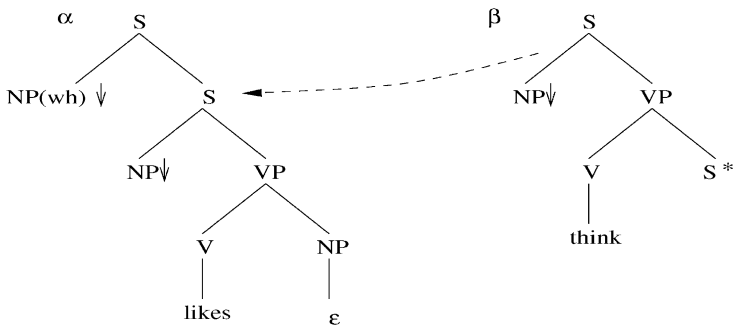



Fig. 15. Wrapping as substitution and adjunction 1.

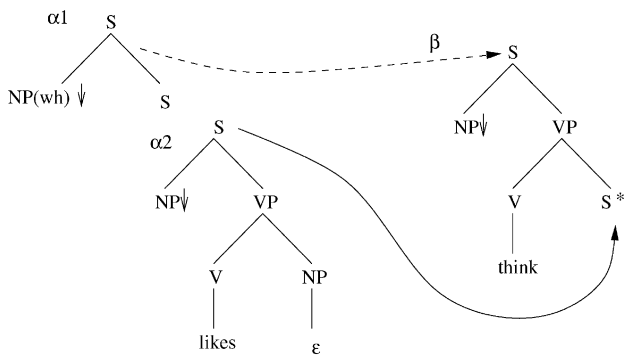



Fig. 16. Wrapping as substitution and adjunction 2.

and 16. The auxiliary tree $\beta$ can be adjoined to the tree $\alpha$ at the indicated node in $\alpha$ as shown in Fig. 15. Alternatively, we can view this composition as adjoining the supertree $\alpha_1$ (the *wh* tree) at the root node of $\beta$ and substitution of the subtree $\alpha_2$ (the *likes* tree) at the foot node of $\beta$ as shown in Fig. 16. The two ways of composing $\alpha$ and $\beta$ are semantically coherent.

The wrapping perspective can be formalized in terms of the so-called multi-component LTAGs (MC-LTAGs). They are called multi-component because the elementary objects can be sets of trees. In our examples, we have two components (in which $\alpha$ was

split). When we deal with multi-components we can violate the locality of the composition very quickly because the different components may be 'attached' (by adjoining or substitution) to different nodes of a tree and these nodes may or may not be part of an elementary tree, depending on whether the tree receiving the multi-component attachments is an elementary or a derived tree. We obtain what are known as tree-local MC-LTAGs if we put the constraint that the tree receiving multi-component attachments must be an elementary tree. It is known that tree-local MC-TAGs are weakly equivalent to LTAGs, however they can give rise to structural descriptions not obtainable by LTAGs, i.e., they are more powerful than LTAG in the sense of strong generative capacity (as characterized by the derivation trees[3]). Thus, the alternate perspective leads to greater strong generative capacity without increasing the weak generative capacity. The whole range of recent works [2, 6, 7, 10] can be seen as attempts to get more SGP from LTAG without going beyond the WGP of LTAG.

### 4.2. Tree insertion grammars (TIG)

LTAGs provide more SGP than CFGs. However, LTAGs have more WGP than CFGs. Schabes and Waters [18] were motivated to find a subclass of LTAGs which is weakly equivalent to CFGs but is powerful enough to lexicalize CFGs and also provide structural descriptions not obtainable by CFGs.

TIGs are similar to LTAGs. They have the same two operations — substitution and adjoining. However, adjoining is limited in the following way. First, in each auxiliary tree the foot node is the leftmost (or rightmost) daughter of the root node. If the foot node is the leftmost daughter of the root node then there can be nodes to its left as long as their yield is empty. That is, modulo an empty string, the foot node is the leftmost daughter of the root node. Similarly, if the foot node is the rightmost daughter of the root node then there can be nodes to its right as long their yield is empty. Secondly, adjoining is allowed only on the right (or left) frontier of the elementary or derived trees.[4] Schabes and Waters [18] have shown that TIGs are weakly equivalent to CFGs. However, TIGs are strong enough to lexicalize CFGs and they can give rise to structural descriptions that are not obtainable by CFGs.

Consider the TIG shown in Fig. 17. Note that the trees of this TIG satisfy the constraints described above. $t$ denotes an empty string. It can be interpreted as the 'trace' of 'a' in the same tree. Now consider a derivation as follows. Starting with $b1$ we adjoin another instance of $b1$ to the first $b1$ at the $S$ node immediately dominating the trace $t$ (which happens to be on the right frontier of $b1$). We now take another instance of $b1$ and adjoin it to the previously derived tree at the $S$ node which is

[3] Weir [22] compares tree-local MC-TAGs and LTAGs and comments that they are weakly equivalent and also strongly equivalent. He is considering the strong equivalence in terms of the derived trees and not in terms of the derivation trees.

[4] Actually, internal adjoining is allowed as long as adjoining by a pair of left and right adjunctions at the same node does not effectively lead to wrapping around the node. If this allowed then we will have a system equivalent to LTAG. We will not need these internal attachments for our current purpose.

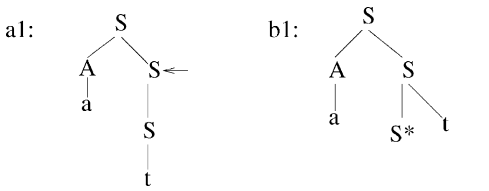



Fig. 17. A TIG for degenerate crossing dependencies.

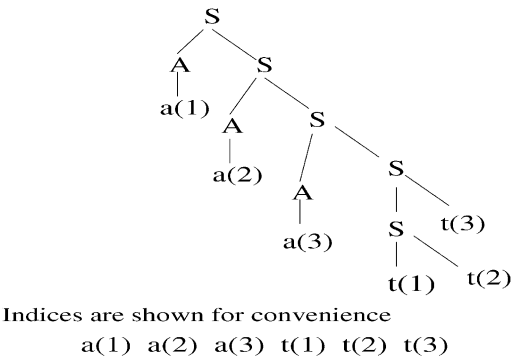
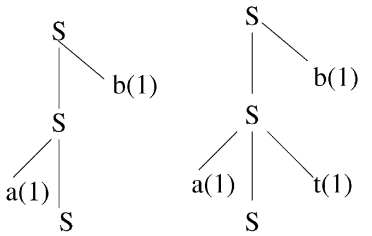



Fig. 18. A derivation in the example TIG.

immediately dominating the rightmost trace $t$. Finally, we adjoin this derived tree to the indicated $S$ node in $a1$. Continuing the derivation in this fashion it is easily seen that the TIG generates a string of $a$'s and $t$'s, where the corresponding (i.e., being in the same elementary tree) $a$'s and $t$'s are indexed for convenience, are in a crossed order as in $a(1)a(2)a(3)\cdots t(1)t(2)t(3)$ (see Fig. 18). The derived tree corresponding to these crosssings is not obtainable by any CFG. Thus TIGs have more SGP than CFGs although they are weakly equivalent CFGs. We will return to this example in the context of Lambek grammars.

## 5. Lambek grammars (LG)

It is well known that the Ajdukiewicz and Bar-Hillel categorial grammars (CG(AB)) are weakly equivalent to CFGss. The derivation trees of CG(AB) are essentially the same as the derivation trees of CFGs. However, for Lambek Grammars (LG) (this is an associative system [11]), the situation is different. In LG, the assignment of categories to lexical items is similar to the assignments in CG(AB) but we have the inference rules associated with the calculus. Although LGs were long conjectured to be weakly equivalent to CFGs it was only relatively recently Pentus [14] proved this conjecture to be true. So now the question arises: Do LGs provide more strong generative power than CFGs? In other words, is it possible to characterize the proof trees of LG in terms of something like the recognizable sets or even beyond recognizable sets? This question




Tree topologies needed for nondegenerate crossing dependencies

Fig. 19. Non-degenerate crossing dependencies.

was originally raised by Buszkowski and van Benthem.[5] Recently, Tiede [21] has investigated this question. He covers a number of aspects and, in particular, shows that the proof trees of LG can be beyond recognizable sets, i.e., there is a Lambek grammar whose proof tree language is not regular. In fact, one of his examples (our example in Fig. 20) can be interpreted as showing that it will be possible to characterize certain kinds of crossing dependencies. We want to point out that Tiede's example is a case where the dependencies are between a lexical item and a lexically empty element. We will call these dependencies as 'degenerate' crossing dependencies, to be distinguished from the case where both elements are lexically non-empty.

We might ask whether it would be possible for the proof trees of a Lambek grammar to characterize true or, non-degenerate crossing dependencies (i.e., when the dependencies are between lexically non-empty elements). We suspect that this would not be possible. If it is possible then it would be indeed quite surprising because, for any known formal system that characterizes crossing dependencies (as between $a$'s and $b$'s in $a^n b^n$) it turns out that the system is weakly more powerful than CFGs. Some examples of such systems are LTAGs, combinatory categorial grammars (CCG) as in [19], linear indexed grammars (LIG) among others. They all generate some context sensitive languages, for example, the language $\{a^n b^n c^n | n \geqslant 1\}$.

The left-hand side tree in Fig. 19 shows the topology needed to obtain the non-degenerate crossing dependencies in LTAG. The derivation proceeds as follows. The left-hand side tree in Fig. 19 is repeatedly adjoined to the $S$ node that is in the center of the spine, starting with the elementary tree and then the derived trees. The resulting derived tree represents crossed dependencies between the corresponding (i.e., belonging to the same elementary tree) $a$'s and $b$'s. The right-hand side tree in Fig. 19 also generates the same language. We interpret the $t$'s as 'traces' or empty elements. It is easily seen that the $a$'s and $t$'s are nested and $t$'s and $b$'s are nested. Thus, the crossing between the $a$'s and $b$'s is the result of two 'coordinated' nested dependencies, coordinated through the empty elements. Note that the empty elements

[5] See the discussion in [1, pp. 688–736].

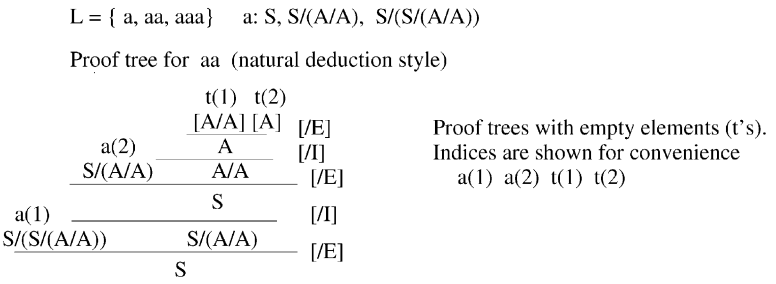



Fig. 20. Degenerate crossing dependencies in a Lambek grammar.

are in the interior and not at the periphery, a crucial point to which we will return in the context of Lambek grammars.

Now, once we have a tree with this topology it is easy to see that the same topology can be used to generate the language $\{a^n b^n c^n | n \geqslant 1\}$. The relevant tree will be the same as the left-hand side tree in Fig. 19 with $a$ and $b$ as left and right daughters of the central $S$ and $c$ as the right daughter of the root node $S$.

Let us now look at Tiede's example (see Fig. 20) of a Lambek derivation where crossing dependencies appear. These crossing dependencies are degenerate, i.e., they are dependencies between pairs, where the first element is a non-empty lexical item and the other element is an empty element. In this example, L denotes the string language, $a$ has been assigned three types, and in the proof tree the $a$'s and $t$'s are indexed for convenience. $E$ and $I$ denote the elimination and introduction rules, respectively.

By suitably arranging the introduction and discharge of assumptions in the hypothetical reasoning in LG we have crossing dependency relations between the $a$'s and the $t$'s, where the $t$'s are the empty elements. These two empty elements $t(1)$ and $t(2)$ correspond to the two assumptions in the proof tree. Each one of these assumptions is then withdrawn using the $[/I]$ rule, the introduction rule. The $[/E]$ rule is used for elimination. Both the assumptions are withdrawn in the deduction, as is required in a natural deduction proof. The assumptions that are introduced and then withdrawn have to appear always at the periphery of the proof tree, as they do so in this example. The dependencies between the $a$'s and $t$'s (corresponding to the assumptions) can be seen as follows. In the second $[/E]$ step in the deduction (second from the top) the category $A/A$ is eliminated in combination with $S/(A/A)$ corresponding to $a(2)$. The category $A/A$ in this step resulted from the withdrawal of the assumption $[A]$ (corresponding to $t(2)$) at the top level. Thus $a(2)$ corresponds to $t(2)$. Similarly $a(1)$ corresponds to $(t1)$. It is easy to see that a natural deduction proof can be constructed for each string in $L$. Thus we have crossing dependencies between the $a$'s and $t$'s.[6]

For true (i.e., non-degenerate) crossing dependencies both the elements have to be non-empty. One of the ways this is accomplished in most systems I know of is to

[6] We talk here of crossing dependencies as represented by the proof tree. Of course, in the corresponding proof net representation the net is planar and there are no crossings, as was correctly noted by a reviewer.

create two sets of nested dependencies, say between $a$'s and $t$'s and between $t$'s and $b$'s, as we have seen before, where the $t$'s are empty elements, as can be seen in the right-hand side tree in Fig. 19. Then the resulting dependencies between the $a$'s and $b$'s become crossed. It is not possible to achieve this in LG because the empty elements have to be at the periphery in the Lambek deduction.

We now return to the tree insertion grammars (TIG) introduced earlier. TIGs are weakly equivalent to CFGs but they are capable of providing structural descriptions not obtainable by CFGs, as we have discussed earlier in Section 4.2. The example in Section 4.2 (see Fig. 18) is exactly the same kind of example as given by Tiede in the context of Lambek deductions. Thus, it is clear that this kind of crossing dependency can be captured in TIGs. It is tantalizing to show that Lambek deductions can be simulated in TIGs. We have not been able to prove this conjecture yet.

## 6. Summary

We discussed some relationships between the strong and week generative powers of formal systems, in particular, from the point of view of squeezing more strong power out of a formal system without increasing its weak power. More specifically, we commented on these issues from the perspectives of context-free grammars, lexicalized tree-adjoining grammars, their two variants (multi-component tree-adjoining grammars and tree insertion grammars), and Lambek grammars (in particular, the associative system). We discussed crossing dependencies (degenerate as well as non-degenerate) and their relationship to the strong generative power. Finally, we suggested a possible relationship between Lambek deductions and the derivations in tree insertion grammars.

## Acknowledgements

I would like to thank the two reviewers of this paper whose valuable comments helped to improve the presentation of this paper.

## References

[1] J. Benthem, A. ter Meulen, in: J. Benthem, A. termeulen (Eds.), Handbook of Logic and Language, MIT Press, Cambridge, MA, 1998, pp. 683–736.
[2] M. Candito, S. Kahane, Defining DTG derivations to get semantic graphs, Proc. TAG+4 Workshop, University of Pennsylvania, August 1998, pp. 25–28.
[3] A.K. Joshi, Tree-adjoining grammars: how much context sensitivity is required to provide reasonable structural descriptions?, in: D. Dowty, L. Karttunen, A. Zwicky (Eds.), Natural Language Parsing, Cambridge University Press, Cambridge, 1985, pp. 206–250.
[4] A.K. Joshi, L. Levy, K. Yueh, Local constraints on transformations, J. Comput. System Sci. 8 (1972) 22–33.
[5] A.K. Joshi, Y. Schabes, Tree-adjoining grammars, in: G. Rosenberg, A. Salomaa (Eds.), Handbook of Formal Languages, Springer, Berlin, 1997, pp. 69–123.

[6] A.K. Joshi, K. Vijay-Shanker, Compositional semantics with lexicalized tree-adjoining grammar (LTAG): how much underspecification is necessary?, in: H.C. Bunt, E.G.C. Thijsse (Eds.), Proc. Third Internat. Workshop on Computational Semantics (IWCS-3) Tilburg, 1999, pp. 131–145.
[7] L. Kallmeyer, A.K. Joshi, Factoring predicate argument and scope semantics: underspecified semantics with LTAG, Proc. Twelfth Amsterdam Colloq., University of Amsterdam, Amsterdam, 1999, pp. 169–174, a revised version will appear in Language and Comput., 2001.
[8] H. Kolb, J. Michaelis, U. Mönnich, An operational and denotational approach to non-context-freeness, Theoret. Comput. Sci. (this Vol.) (2003).
[9] A. Kroch, A.K. Joshi, Linguistic relevance of tree-adjoining grammars, Tech. Report, Department of Computer and Information Science, University of Pennsylvania, 1985.
[10] S. Kulick, Constrained non-locality: long-distance dependencies in TAG, Ph.D. Dissertation, University of Pennsylvania, Philadelphia, USA, 2000.
[11] J. Lambek, The mathematics of sentence structure, Amer. Math. Monthly 65 (1958) 154–169.
[12] J.W. McCawley, Concerning the base component of a transformational grammar, Found. Language 4 (1967) 55–81.
[13] P.M. Miller, Strong Generative Capacity, CSLI Publications, Stanford University, Stanford CA, 1999.
[14] M. Pentus, Lambek grammars are context-free, Proc. 8th Ann. Symp. on Logic in Computer Science, 1993.
[15] S. Peters, R. Ritchie, Context sensitive immediate constituent analysis: context-free languages revisited, ACM Symp. on Theory of Computing, Marina del Rey, 1969, pp. 1–8.
[16] J. Rogers, Grammarless phrase structure grammar, Linguistics Philos. 20 (1997).
[17] Y. Schabes, S, Shieber, An alternative conception of tree-adjoining derivation, Computat. Linguistics 20 (1) (1994) 91–124.
[18] Y. Schabes, R.C. Waters, Tree insertion grammars, Tech. Report TR-94-13, Mitsubishi Electric Research Laboratories, Cambridge, 1994.
[19] M. Steedman, Surface Structure and Interpretation, MIT Press, Cambridge, MA, 1996.
[20] J.W. Thatcher, Characterizing derivation trees of context-free grammars through a generalization of finite automata theory, J. Comput. System Sci. 1 (1967) 317–322.
[21] H. Tiede, Deductive systems and grammars, Ph.D. Dissertation, Indiana University, Bloomington, 1999.
[22] D. Weir, Characterizing mildly context-sensitive grammar formalisms, Ph.D. Dissertation, University of Pennsylvania, Philadelphia, 1988.